Ordonnancement temps réel et minimisation de la consommation d'énergie

Nicolas Navet, Bruno Gaujal

Chapitre 4

Ordonnancement temps réel et minimisation de la consommation d'énergie

4.1. Introduction

La consommation en énergie est devenue un problème crucial dans la conception des équipements électroniques dont l'alimentation est assurée par des batteries. Parmi tous les composants électroniques, le processeur est particulièrement utilisateurs d'énergie puisque des études ([POU 01, ZEN 02] cités dans [AYD 04]) ont montré qu'il pouvait à lui seul utiliser plus de 50 % de l'énergie lorsqu'il était sollicité intensivement. En jouant sur une réduction de la fréquence de fonctionnement du processeur, des stratégies d'ordonnancement adaptées permettent de réduire considérablement la consommation énergétique. Nous proposons dans ce chapitre un tour d'horizon, dans le cas mono-processeur, des techniques d'ordonnancement visant à minimiser la consommation d'énergie tout en garantissant le respect de contraintes d'échéances. Les contraintes de temps peuvent peser explicitement sur certaines activités du système ou peuvent provenir de contraintes de performances minimales si l'on sort du cadre classique des systèmes temps réel. Le problème d'ordonnancement à résoudre consiste non seulement à déterminer l'ordre dans lequel exécuter les activités du système mais également à fixer la fréquence de fonctionnement du processeur au cours du temps. Comme souligné dans [GRU 02], l'ordonnancement sous contrainte d'énergie acquiert une nouvelle dimension qui est la vitesse du processeur.

Chapitre rédigé par Nicolas NAVET et Bruno GAUJAL.

Nous expliquerons en premier lieu le contexte de l'ordonnancement sous contraintes de temps et d'énergie, situerons les stratégies d'ordonnancement par rapport à l'ensemble des techniques visant à réduire la consommation et définirons les contraintes de temps et les modèles d'activités qui seront considérées par la suite.

4.1.1. *Contexte du problème*

L'autonomie est un problème majeur pour de nombreux équipements électroniques dont l'alimentation en énergie est assurée par des batteries, comme par exemple des ordinateurs portables, des assistants intelligents, des robots mobiles, des implants médicaux (*pacemaker*) ou des téléphones portables. L'augmentation continue des performances et des fonctionnalités de ces équipements nécessite l'utilisation de composants électroniques fonctionnant à des fréquences toujours plus élevées et donc consommant de plus en plus d'énergie. Ainsi, par exemple, un processeur de type Intel P4 Mobile 1.8GHz consomme au plus 30W [GRU 02] contre 3W pour un Intel I386 [ANC 03]. Parallèlement, la technologie des batteries ne progresse pas suffisamment vite pour satisfaire ces nouveaux besoins. Il est donc nécessaire de travailler à la réduction de la consommation en énergie d'autant plus que le gain se situe non seulement au niveau de l'autonomie des systèmes, mais également au niveau de la place et du poids consacrés aux batteries ainsi qu'au niveau des dissipations thermiques et donc de la fiabilité de l'électronique.

4.1.2. *Techniques de réduction de la consommation*

Dans ce paragraphe, nous faisons un tour d'horizon des techniques de réduction de la consommation en énergie, le lecteur pourra se référer à [PAR 00] pour un développement plus complet sur ce sujet.

Une première stratégie est de travailler sur la technologie des composants matériels. Ainsi une diminution de la taille des composants, rendue possible par des progrès dans les techniques de fabrication, permet une tension d'alimentation plus faible et donc une consommation moindre. Une seconde stratégie est de limiter l'alimentation d'un composant aux blocs nécessaires pour le traitement en cours, par exemple, on peut diviser une mémoire cache en des blocs pouvant être activés indépendamment les uns des autres. Une autre possibilité est de limiter le nombre de changements d'états dans un circuit car chaque changement d'état induit un coût énergétique. Ainsi dans [SU 95], il a été proposé d'utiliser pour l'adressage mémoire le codage Gray qui garantit un seul bit de différence entre un nombre et son successeur contrairement au classique codage en complément à deux. Il est également possible d'intervenir au niveau de l'architecture matériel, par exemple en remplaçant un disque dur, gourmand en énergie du fait de son système mécanique, par une mémoire flash. Le dimensionnement de l'architecture matériel est également important, ainsi il s'agit de

trouver la bonne taille pour la ou les mémoires caches sachant que plus elles sont de grande taille plus elles consomment mais moins des accès coûteux vers d'autres mémoires seront nécessaires. Une autre voie qui est explorée est de spécialiser les composants pour l'usage fait par exemple à l'aide de composants reconfigurables FPGA [ABN 98, DAV 03]. Enfin, certains travaux étudient l'utilisation de circuits électroniques asynchrones qui, contrairement aux circuits synchrones traditionnels, présentent la caractéristique intéressante de ne consommer de l'énergie que dans les sous-parties du circuit réellement utilisées lors de l'exécution d'une instruction (voir [REM 03] pour un état de l'art).

Le logiciel a également un rôle important à jouer pour minimiser la consommation en énergie, par exemple, en optimisant le code des programmes exécutables ; ainsi remplacer des opérations de mémoire-à-mémoire par des opérations de registre-à-registre apportent des gains substantiels [TIW 94]. Il peut être aussi parfois bénéfique d'effectuer de l'expansion en-ligne (*inlining*) pour limiter le nombre d'appels de fonctions, qui est une opération souvent longue et coûteuse ; la contrepartie étant que le code grossit nécessairement et qu'il puisse alors ne plus être contenu dans une mémoire cache.

Il existe naturellement des techniques dites hybrides basées sur une collaboration entre composants matériels et logiciels. Par exemple des stratégies de mise en veille plus ou moins profondes de composants (voir paragraphe 4.2.2.1), ou d'adaptation du voltage du processeur, et donc de la fréquence, au besoin courant de l'application en termes de performances (voir paragraphe 4.1.3). Cette dernière classe de techniques permet des réductions de consommation importantes car, sans considérer la puissance statique, l'énergie consommée varie au minimum en le carré du voltage dans les technologies CMOS actuelles. Les résultats obtenus avec ces techniques d'adaptation du voltage, qui ont constitué au cours des dix dernières années une thématique de recherche importante dans la communauté de l'économie d'énergie, sont le sujet de cet article.

4.1.3. *Contraintes de temps et d'énergie*

L'apparition de composants électroniques à tension d'alimentation variable constitue un progrès majeur dans l'optique d'une plus grande autonomie et, à l'heure actuelle, de nombreux processeurs comportant cette possibilité sont disponibles commercialement. On peut citer par exemple les processeurs de la famille Crusoe de la société Transmeta, la technologie PowerNow ! d'AMD ou les technologies SpeedStep et XScale d'Intel (pour plus de détails, se référer à [GRU 02]). La puissance dynamique dissipée par un composant variant au minimum en le cube de la fréquence de fonctionnement (voir le paragraphe 4.2.1), il est judicieux de faire fonctionner le processeur (CPU pour « *Central Processing Unit* » dans la suite) à la fréquence la plus faible compatible avec le niveau de performance requis.

Ainsi, lorsque des contraintes de temps explicites pèsent sur certaines activités du système, il s'agit naturellement de les respecter avec l'objectif supplémentaire de minimiser la consommation en énergie. Le problème d'ordonnancement à résoudre consiste non seulement à déterminer l'ordre dans lequel exécuter les activités du système mais également à fixer la fréquence de fonctionnement du processeur au cours du temps. Comme souligné dans [GRU 02], l'ordonnancement sous contrainte d'énergie acquiert une nouvelle dimension qui est la vitesse du processeur.

Si aucune contrainte de temps n'est spécifiée alors la meilleure stratégie vis-à-vis de la consommation est de mettre le processeur en veille ce qui naturellement est incompatible avec le niveau de performances minimales attendu. Une technique possible pour garantir le bon fonctionnement du système est alors d'allouer à chacune des activités une date d'échéance, et l'on voit que le problème de l'ordonnancement sous contraintes de temps et d'énergie a des applications en dehors du cadre classique des systèmes temps réel (voir par exemple [LOR 01]). Une autre possibilité pour obtenir les performances minimales est de ne pas considérer des échéances individuelles mais de raisonner en termes de nombre minimal de requêtes traitées par unité de temps *(throughput*) ou de nombre maximum de requêtes en attente de traitement. Ces dernières possibilités ne seront pas explorées dans le cadre de cet article qui ne traite que de l'ordonnancement avec contraintes d'échéances explicites.

4.1.4. *Modèles de tâches et notations*

Nous considérons deux types de tâches s'exécutant sur une plate-forme monoprocesseur : des tâches récurrentes, en pratique généralement périodiques, et des tâches non récurrentes appelées aussi « *jobs* » ou « tâches apériodiques » dans la littérature.

Les tâches récurrentes du système constituent un ensemble fini noté $\mathcal{T} = \{\tau_1, \ldots, \tau_m\}$ de cardinalité m où τ_k est la tâche d'indice k dont la $n^{\text{ième}}$ instance, notée $\tau_{k,n}$, possède certaines caractéristiques indépendantes de l'ordonnancement qui sera réalisé :

– son instant de mise à disposition $A_{k,n}$ (ou date d'arrivée) avec $0 \leq A_{k,1} \leq \ldots \leq A_{k,n-1} \leq A_{k,n} \leq \ldots$. C'est le premier instant à partir duquel l'instance $\tau_{k,n}$ est susceptible d'être exécutée ;

– son pire temps d'exécution (*Worst-Case Execution Time* - WCET) sur le processeur à sa fréquence maximale, noté $C_{k,n}$. Les stratégies d'ordonnancement consistant à réduire la fréquence de fonctionnement, le temps d'exécution effectif sera donc généralement supérieur à $C_{k,n}$. Dans la suite, nous considérerons le cas le plus courant où toutes les instances d'une même tâche ont le même temps d'exécution que nous noterons C_k ;

– son échéance $D_{k,n}$, c'est à dire l'instant auquel $\tau_{k,n}$ doit avoir fini de s'exécuter. L'échéance relativement à la date d'arrivée est $\overline{D}_{k,n}$ avec $\overline{D}_{k,n} = D_{k,n} - A_{k,n}$. Dans

la suite, toutes les instances d'une même tâche τ_k seront supposées avoir la même échéance relative notée $\overline{D}_k$;

– le temps entre la date d'arrivée de $\tau_{k,n}$ et la date d'arrivée de la prochaine instance $\tau_{k,n+1}$, appelé temps interarrivées ou temps de cycle, est noté $T_{k,n}$ avec $T_{k,n} \stackrel{\text{def}}{=} A_{k,n+1} - A_{k,n}$. Si les temps interarrivées sont identiques ($T_{k,n} = T_k \ \forall n$), la tâche τ_k est strictement périodique.

Les activités non récurrentes forment un ensemble noté $\mathcal{T} = \{\tau_1, \ldots, \tau_m\}$ où τ_k est la tâche d'indice k de temps d'exécution C_k à vitesse maximale, de date d'arrivée A_k et d'échéance D_k.

4.1.5. *Objectifs et organisation*

L'objectif de cet article est en premier lieu d'expliquer la problématique de l'ordonnancement sous contraintes de temps et d'énergie puis de présenter les résultats de base qui existent dans le domaine. Compte tenu de l'abondance de la littérature, il est impossible de prétendre à un état de l'art exhaustif dans le format imparti ; la solution de présenter uniquement une taxinomie détaillée des approches n'a pas été non plus retenue pour ne pas masquer la réalité des techniques utilisées et des problèmes à résoudre. Nous avons choisi de ne traiter que des deux politiques d'ordonnancement les plus importantes du temps réel que sont EDF (*Earliest Deadline First*) et FPP (*Fixed Priority Preemptive*). Le lecteur est renvoyé au chapitre 1 de ce livre pour plus de détails sur ces politiques. Nous considérerons le modèle de tâches le plus simple, c'est à dire des tâches indépendantes les unes des autres, mais le lecteur pourra trouver des pointeurs vers des travaux traitant de modèles plus généraux comme les tâches DAG (*Directed Acyclic Graph*, par exemple dans [RAO 06]) ou modélisant plus finement les interactions avec le matériel (voir, par exemple, [BIN 05]).

Nous distinguons deux grandes classes de stratégies. Tout d'abord, les politiques pour lesquelles le choix de la vitesse de fonctionnement à un instant donné est indépendant de l'historique de l'ordonnancement. Les choix sont donc effectués avant l'exécution de l'application et l'on qualifie ces techniques de hors-ligne. Dans cette première catégorie, on peut effectuer une distinction entre les algorithmes qui calculent une vitesse de fonctionnement unique pour l'ensemble du système pendant toute sa durée de vie (voir paragraphe 4.3.1) et ceux qui individualisent la vitesse en fonction de la tâche ou de l'instance (voir paragraphe 4.3.2). Ces derniers sont naturellement a priori plus efficaces mais présupposent la capacité de modifier dynamiquement la fréquence du processeur (voir paragraphe 4.2.2).

La seconde grande classe de techniques, présentée en section 4.4, comprend les politiques d'ordonnancement qui utilisent en-ligne des informations sur l'état du système pour fonder leurs choix quant aux fréquences. En particulier, ces techniques sont les

plus efficaces d'un point de vue énergétique lorsque le temps d'exécution des tâches est inférieur à leur WCET, comme c'est le plus souvent le cas dans les systèmes temps réel.

Nous conclurons en section 4.5 en comparant les différentes approches envisagées puis, en section 4.6, identifierons les directions de recherche que nous jugeons les plus prometteuses en ordonnancement faible consommation.

4.2. Consommation énergétique d'un processeur

Dans cette section, nous expliquons quelques points-clés portant sur la consommation en énergie des processeurs. Nous examinons ensuite les différentes technologies de processeurs permettant une réduction de la consommation.

4.2.1. *Puissance dissipée et consommation énergétique*

Les concepts de cette section sont valables pour tout circuit CMOS (*Complementary Metal Oxide Semiconductor*) qui est la technologie dominante dans les circuits électroniques. Le lecteur désirant des développements plus approfondis sur ce sujet pourra consulter [GRU 02, SHI 00a].

L'énergie consommée dans un intervalle de temps $[a, b]$ est par définition l'intégrale de la puissance dissipée $E = \int_a^b P(t)dt$ où $P(t)$ est la puissance dissipée à l'instant t. Cette puissance dissipée dans un circuit électronique se compose de la puissance statique et de la puissance dynamique. Dans les circuits CMOS la puissance dynamique représente de l'ordre de 80-85 % de la puissance dissipée et, classiquement, on néglige la puissance statique[1]. La puissance dissipée totale peut donc s'exprimer par :

$$P \approx P_{dynamique} \sim \alpha f C V^2 \qquad [4.1]$$

où α est le nombre de transitions par cycle d'horloge, f est la fréquence de fonctionnement, C est la capacité équivalente et V est la tension d'alimentation. On voit dans [4.1] qu'il existe quatre paramètres pour diminuer l'énergie consommée et toutes les techniques de réduction de la puissance dynamique s'attaquent à l'un ou l'autre de ces facteurs. Le terme α dépend des données traitées et de la technique de codage utilisée, C est une caractéristique du circuit utilisé. Réduire la fréquence f sans modifier la tension sera sans effet au niveau de la consommation car, globalement, le temps nécessaire pour terminer une même séquence de code augmente d'un facteur k si l'on

1. Comme cela sera discuté en section 4.6, avec l'évolution des technologies des semiconducteurs, cette hypothèse devra être levée et c'est une perspective de recherche importante en ordonnancement faible consommation.

réduit la fréquence d'un même facteur k. Il est finalement possible de diminuer la tension V mais fréquence et voltage sont liés par la relation :

$$\frac{1}{f} \sim \frac{V}{(V - V_t)^\gamma}$$

avec V_t la tension de seuil et γ une constante. Pour une tension de seuil suffisamment petite par rapport à la tension d'alimentation, la relation entre fréquence et tension d'alimentation devient $f \sim V^{\gamma-1}$. Dans le modèle MOSFET (*Metal Oxide Semiconductor Field Effect Transistor*) classique, γ est approximé par deux ; la fréquence est donc linéaire en la tension et la puissance varie en le cube de la fréquence. Certains autres modèles considèrent des valeurs différentes de γ (par exemple $\gamma = 1,3$ dans [RAB 96] cité dans [GRU 02]) mais, en pratique, il n'est pas crucial de déterminer l'expression exacte de la puissance car la puissance dynamique dissipée reste toujours une fonction convexe croissante de la fréquence et beaucoup de résultats énoncés en économie d'énergie sont valables pour toute fonction convexe croissante. Le lecteur pourra consulter [GAU 05b] pour une illustration dans le cas de résultats portant sur EDF.

Pour une tension d'alimentation donnée, il existe une fréquence de fonctionnement optimale du point de vue énergétique qui est la fréquence maximale supportée par le circuit à cette tension. Dans la suite, plutôt que de raisonner en termes de fréquence, nous parlerons de la vitesse du processeur qui est le rapport entre la fréquence de fonctionnement courante et la fréquence maximale du processeur, appelée aussi fréquence nominale.

4.2.2. *Technologie des processeurs*

Une fraction importante des microprocesseurs disponibles commercialement sont conçus dès l'origine dans l'optique d'une faible consommation. Dans [ANC 03], l'auteur chiffre la réduction de consommation d'énergie par rapport à des processeurs classiques comme étant de l'ordre d'un facteur 10 pour une réduction de puissance d'un facteur 2 à 3. Nous distinguons deux classes de processeurs en fonction de la possibilité ou non de changer la fréquence nominale de fonctionnement ; l'efficacité des stratégies d'ordonnancement « économes en énergie » sera naturellement dépendante de cette caractéristique du processeur.

4.2.2.1. *Processeurs à vitesse constante et mode veille*

Les processeurs à vitesse constante opèrent à leur fréquence d'horloge et leur tension d'alimentation nominales et consomment donc la même quantité d'énergie à l'exécution (modulo le fait que toutes les instructions processeurs ne consomment pas exactement la même quantité d'énergie). Le plus souvent, ces processeurs possèdent au minimum deux modes de fonctionnement, le mode actif et le mode veille

dans lequel aucune instruction n'est exécutée avec une consommation énergétique grandement réduite. Ainsi, le processeur Intel 80200 [Int 03] possède trois modes de fonctionnement, dont deux modes faible consommation qui diffèrent par le nombre de modules du processeur mis en veille, la consommation en mode veille et les temps de remise en fonctionnement. Les techniques qui visent à sélectionner au mieux les modes de fonctionnement des ressources sont connues sous le terme de *Dynamic Power Management* (DPM), et certaines sont par exemple implantées dans le standard ACPI (*Advanced Configuration and Power Interface*, voir [COM 02]). Le lecteur pourra consulter [BEN 00] pour un état de l'art sur les approches DPM.

4.2.2.2. *Processeurs à vitesse variable*

Des processeurs plus spécifiquement conçus pour l'économie d'énergie permettent de varier la tension d'alimentation et donc la fréquence de fonctionnement. Les stratégies d'adaptation dynamique de la tension sont connues sous le terme de *Dynamic Voltage Scaling* (DVS). Remarquons que même si dans la littérature l'hypothèse d'une plage de fréquence continue est souvent faite, la technologie actuelle des processeurs synchrones implique nécessairement un nombre de fréquences fini.

Parmi les processeurs à fréquence variable, on peut distinguer ceux qui permettent un changement de fréquence pendant l'exécution d'une application et ceux qui ne le permettent pas (nécessité de ré-initialisation, temps de changement de fréquence trop importants, etc.). On peut citer parmi les processeurs à vitesse variable, les processeurs Transmeta Crusoe, le lpARM (UC Berkeley) et le processeur Intel Pentium 4M. Le lecteur pourra consulter [GRU 02] et [SAL 03] pour plus de détails.

4.3. Politiques hors-ligne

Les politiques hors-ligne utilisent les informations disponibles avant l'exécution de l'application (c'est-à-dire politique d'ordonnancement et caractéristiques des tâches) pour en dériver les vitesses de fonctionnement. Les vitesses utilisées à l'exécution ne dépendent alors pas de l'état du système. Nous distinguons le cas où l'on recherche une vitesse unique pour toutes les tâches du système (paragraphe 4.3.1) et le cas où l'on peut individualiser les vitesses en fonction de l'instance en cours d'exécution (paragraphe 4.3.2). Le cas une vitesse par tâche, qui présente également un intérêt en pratique car il offre un bon compromis entre surcharge à l'exécution / difficultés d'implémentation et performances, ne peut être traité dans le format imparti mais le lecteur trouvera des pointeurs dans les paragraphes 4.3.1 et 4.3.2.

4.3.1. *Vitesse unique pour le système*

Les politiques à fréquence unique sont parfois référencées dans la littérature sous l'appellation de méthodes MRS (*Minimum Required Speed*). Ce sont des approches

hors-ligne dont l'objectif est de déterminer une vitesse unique du processeur valable pendant toute la durée de vie de l'application.

Comme il n'y a pas de sélection en-ligne des fréquences, ces stratégies sont utilisables sans aucune modification au niveau du système d'exploitation et elles n'induisent pas d'*overhead* à l'exécution. D'autre part, ces stratégies sont les seules compatibles avec des processeurs qui ne peuvent changer de fréquence que hors-ligne (voir le paragraphe 4.2.2.2). Logiquement, ces techniques sont d'une façon générale moins efficaces que celles qui permettent de changer dynamiquement la fréquence par exemple en fonction de la charge courante du système [WEI 94] ou de la tâche à exécuter (voir les paragraphes 4.3.2.1 et 4.3.2.2). En effet, la vitesse du système sera celle permettant l'exécution de l'activité la plus contrainte.

Seront présentés dans cette section les résultats de base existants pour EDF (*Earliest Deadline First*, voir chapitre 1) et FPP (*Fixed Priority Preemptive*, voir chapitre 1) pour des tâches indépendantes (c'est-à-dire pas de relations de précédence entre tâches et aucune ressource partagée). D'autres modèles de tâches et d'autres politiques (voir par exemple Round-Robin dans [BRI 04]) ont déjà été largement étudiés dans ce même contexte des politiques à fréquence de fonctionnement unique.

4.3.1.1. *Ordonnancement EDF*

Des tests de faisabilité, qui ne nécessitent pas de calcul explicite de temps de réponse, existent pour la politique EDF et ceux-ci couvrent les contextes d'utilisation les plus courants. Nous verrons que la vitesse minimale du système, notée $\mathcal{S}_{\text{edf}}$, peut être souvent calculée à l'aide de ces tests de faisabilité.

4.3.1.1.1. Tâches périodiques avec échéances égales aux périodes

Lui et Layland [LIU 73] ont montré qu'un ensemble de tâches périodiques synchrones (c'est-à-dire toutes mises à disposition simultanément) à échéances sur requêtes ($\forall k,\ \overline{D}_k = T_k$) était faisable si et seulement si (ssi) la charge du système $U_{\mathcal{T}}$ vérifiait :

$$U_{\mathcal{T}} = \sum_{\tau_i \in \mathcal{T}} \frac{C_i}{T_i} \leqslant 1.$$

Ce résultat est également valable pour des tâches asynchrones comme cela a été montré dans [COF 76]. Si la charge est inférieure à 1, il est possible de diminuer la vitesse du CPU et ce, au maximum, jusqu'à une vitesse qui conduise à un taux d'utilisation de 1, c'est à dire simplement $\mathcal{S}_{\text{edf}} = U_{\mathcal{T}}$. Dans ce cas particulier de tâches à échéances sur requêtes sous EDF, il n'est pas possible de trouver une meilleure solution que celle-ci même en autorisant différentes vitesses de fonctionnement. Le lecteur pourra se référer à [AYD 01] pour plus de détails.

4.3.1.1.2. Tâches périodiques avec échéances inférieures aux périodes

Pour des tâches périodiques à échéances inférieures à la période (voir [STA 98]), une condition suffisante d'ordonnançabilité est :

$$\alpha_{\mathcal{T}} = \sum_{\tau_i \in \mathcal{T}} \frac{C_i}{\min(D_i, T_i)} \leqslant 1. \qquad [4.2]$$

On peut donc permettre des vitesses processeurs inférieures à la vitesse nominale tant que cette condition est vérifiée ; la vitesse minimale qui satisfait [4.2] est $\mathcal{S}_{\text{edf}} = \alpha_{\mathcal{T}}$. Comme la condition n'est que suffisante et non nécessaire, il peut parfaitement exister des vitesses inférieures à $\mathcal{S}_{\text{edf}} = \alpha_{\mathcal{T}}$ qui permettent la faisabilité et des gains en énergie supérieurs.

Une alternative ne comportant pas cet inconvénient est de calculer la vitesse minimale requise pour chacune des instances pendant une "période" du système (i.e. 1 ppcm des périodes pour des tâches synchrones, 2 ppcm + $\max\{A\}$ dans le cas asynchrones) à l'aide des techniques discutées au paragraphe 4.3.2.1 et de considérer le maximum. Notons que cette même analyse reste valable dans le cas de tâches à échéances plus grandes que la période.

4.3.1.2. *Ordonnancement FPP*

Nous présentons ici deux techniques simples, dérivées de tests de faisabilité, qui permettent de trouver une vitesse unique pour ordonnancer un ensemble de tâches sous FPP.

4.3.1.2.1. Test de faisabilité basé sur la charge

Dans le cas de tâches périodiques à échéances sur requête, il est possible de se servir du test de faisabilité de Lui et Layland [LIU 73] pour déterminer une vitesse minimale unique pour un ensemble de tâches $\mathcal{T}$. On sait que $\mathcal{T}$ est nécessairement faisable sous FPP si la charge du processeur $U_{\mathcal{T}}$ vérifie $U_{\mathcal{T}} = \sum_{\tau_i \in \mathcal{T}} \frac{C_i}{T_i} \leq m(2^{\frac{1}{m}} - 1)$. Si tel est le cas, on peut alors fixer la vitesse minimale du système à :

$$\mathcal{S}_{\text{fpp}} = \frac{U_{\mathcal{T}}}{m(2^{\frac{1}{m}} - 1)}.$$

Si elle est simple, cette approche à deux inconvénients. La condition de Lui et Layland n'étant qu'une condition suffisante et non une condition nécessaire, il serait parfaitement possible pour certaines configurations de réduire la vitesse en deçà de $U_{\mathcal{T}}/m(2^{\frac{1}{m}} - 1)$ tout en gardant la faisabilité. D'autre part, cette approche n'est utilisable que dans le cadre du modèle de tâches périodiques à échéances égales à la période.

4.3.1.2.2. Analyse exacte de la faisabilité

Lehoczki *et alli* ont présenté dans [LEH 89] une condition nécessaire et suffisante pour tester la faisabilité d'un ensemble de tâches périodiques à échéances inférieures ou égales aux périodes. Dans [SHI 00a], les auteurs utilisent ce test pour dériver une vitesse de fonctionnement minimale sous FPP.

La fonction $W_i(t) = \sum_{j=1}^{i} C_j \cdot \lceil \frac{t}{T_j} \rceil$ est la charge de travail soumise par l'ensemble des tâches $\tau_1, \ldots, \tau_i$ au cours du temps. Le théorème 1 de [LEH 89] dit que s'il existe un instant t tel que $W_i(t) \leq t$ pour $t \leq T_i$ alors τ_i est ordonnançable. Il est également montré qu'il suffit de vérifier un nombre fini d'instants t, appelés des points d'ordonnancements. Pour une tâche τ_i à échéance égale à la période, cet ensemble est :

$$S_i = \{k \cdot T_j \,|\, j = 1, \ldots, i; k = 1, \ldots, \lfloor T_i/T_j \rfloor\}. \quad [4.3]$$

Si l'échéance est inférieure à la période l'ensemble est $S_i = \{t \,|\, t \in S_i \wedge t < D_i\} \cup \{D_i\}$ avec S_i initialement calculé selon [4.3] (voir [GRU 02]). On note $S_{i,j}$ le $j$ème élément de S_i avec S_i trié par ordre croissant et $\eta_{i,j}$ est le facteur de réduction de vitesse tel que :

$$\frac{1}{\eta_{i,j}} W_k(S_{i,j}) = S_{i,j}. \quad [4.4]$$

Avec la vitesse $1/\eta_{i,j}$, τ_i est ordonnançable grâce au point d'ordonnancement $S_{i,j}$ (voir théorème 1 de [LEH 89]). Comme il suffit d'un point d'ordonnancement vérifiant [4.4], le plus grand facteur de réduction admissible pour τ_i est $\eta_i = \max_j \eta_{i,j}$. Toutes les tâches devant être faisables, la vitesse minimale requise pour l'ensemble du système est :

$$\mathcal{S}_{\text{fpp}} = 1/\min_{i \in \mathcal{T}} \eta_i.$$

Notons que ce test de faisabilité basé sur les points d'ordonnancement a été utilisé dans [GRU 02] et [SAE 03] pour trouver des solutions heuristiques dans le cas où l'on autorise une fréquence de fonctionnement par tâche, et non plus une fréquence pour le système tout entier. Une solution optimale au cas une vitesse par tâche peut être trouvée en prenant pour vitesse le maximum requis sur l'ensemble des instances d'une tâche avec l'algorithme proposé dans [QUA 02]. Néanmoins, la complexité de l'algorithme (voir paragraphe 4.3.2.2) restreint cette approche à de petits ensembles de tâches.

4.3.2. *Vitesse unique par instance de tâche*

Nous présentons ici des techniques qui visent à minimiser la fréquence de chacune des instances des tâches sous leur hypothèse de WCET. Ce sont les techniques hors-ligne les plus efficaces théoriquement mais dont l'utilisation peut être parfois problématique compte tenu du nombre d'instances dans une période du système et donc, de

l'espace mémoire nécessaire pour stocker les fréquences du CPU. Néanmoins, ce sont des techniques de base souvent utiles dans la résolution des problèmes de type une vitesse par tâche ou même une vitesse pour le système lorsqu'il n'y a pas de résultats optimaux pour le modèle de tâches considéré. C'est le cas en général pour les tâches à échéances plus grandes que la période ou des tâches qui ont des *patterns* d'activation complexes. Notons que les processeurs compatibles avec les techniques présentées dans ce paragraphe doivent être capables de changer de fréquence en-ligne (voir le paragraphe 4.2.2.2).

4.3.2.1. *Ordonnancement EDF*

Dans [YAO 95], les auteurs proposent un algorithme qui calcule les fréquences de fonctionnement optimales du point de vue énergie pour un ensemble de tâches non récurrentes (appelées aussi *jobs*) à échéances. La politique d'ordonnancement sous-jacente étant EDF, cette stratégie est également optimale d'un point de vue ordonnançabilité. C'est un résultat de base en ordonnancement faible consommation qui a inspiré de nombreux travaux ultérieurs comme ceux présentés dans le paragraphe 4.3.2.2.

L'algorithme de Yao *et alli* commence par identifier l'intervalle de temps, appelé « intervalle critique », sur lequel la vitesse de fonctionnement maximum du processeur est requise. L'intensité d'un intervalle est défini comme la charge des *jobs* appartenant à cet intervalle divisé par la durée de l'intervalle, où l'on dit qu'un *job* « appartient » à un intervalle si sa date d'arrivée et sa date d'échéance se situent à l'intérieur de l'intervalle. Intuitivement, l'intensité est la plus petite quantité de travail qui doit être faite dans l'intervalle pour respecter les échéances. L'intensité sur un intervalle $[a, d]$ est donc $W_{[a,d]} = \sum_{A_i \geq a \wedge D_i \leq d} C_i/(d-a)$.

La vitesse minimale admissible sur l'intervalle critique est assignée aux tâches qui appartiennent à cet intervalle. On construit ensuite un nouveau problème en supprimant l'intervalle déjà étudié et l'on détermine le prochain intervalle critique. On peut montrer qu'un intervalle critique commence toujours par une date d'arrivée et termine par une date d'échéance. La figure 4.1 illustre le processus de suppression d'un intervalle de temps $[A_k, D_k]$ qui modifie l'ensemble des tâches de la façon suivante :

– si $A_i \geq A_k$ et $D_i \leq D_k$ (τ_i appartient à l'intervalle) alors τ_i est supprimée et sa fréquence est fixée à l'intensité de l'intervalle $W_{[A_k,D_k]}$;

– si $A_i \in [A_k, D_k]$ alors $A_i := A_k$, sinon si $A_i \geq D_k$ alors $A_i := A_i - (D_k - A_k)$;

– si $D_i \in [A_k, D_k]$ alors $D_i := A_k$, sinon si $D_i \geq D_k$ alors $D_i := D_i - (D_k - A_k)$.

Comme il y a au plus m intervalles critiques successifs (un *job* par intervalle critique) et que déterminer l'intervalle critique est quadratique en le nombre de tâches (il y a m dates d'arrivée et m dates d'échéances), la complexité de l'algorithme est

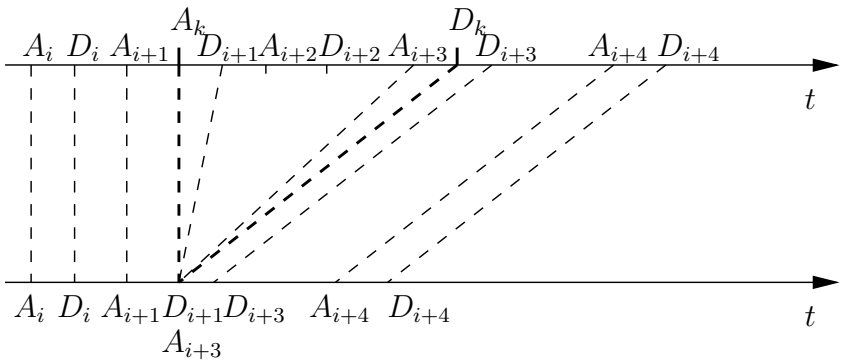

Figure 4.1. *Exemple de suppression de l'intervalle critique $[A_k, D_k]$; les tâches τ_k et τ_{i+2}, qui appartiennent toutes deux à l'intervalle, voient leur fréquence fixée à la valeur de l'intensité requise sur l'intervalle et sont supprimées de l'ensemble des tâches encore à examiner*

$O(m^3)$. Remarquons que cet algorithme fournit un test d'ordonnançabilité sous EDF car un ensemble de tâches non récurrentes est faisable ssi la charge sur tous les intervalles critiques est inférieure à 1. Ce résultat a été redécouvert indépendamment par Spuri, voir théorème 3.5 dans [STA 98].

Récemment, une nouvelle approche de résolution, dans laquelle ce même problème d'ordonnancement est envisagé sous le jour d'un problème de géométrie de type « plus court chemin », a été proposée dans [GAU 05b, GAU 05a]. Le premier intérêt est de réduire la complexité de calcul : $O(m^2 \log(m))$ pour la complexité dans le cas moyen contre $O(m^3)$ pour l'algorithme de Yao *et alli*. Si cela n'apparaît pas spectaculaire à première vue, en pratique, sur des ensembles de 9000 tâches, on observe une réduction du temps de calcul d'un facteur supérieur à 40 [GAU 05a]. Dans le cas particulier de tâches FIFO ($A_i \leq A_j \rightarrow D_i \leq D_j$, la complexité pire-cas de l'approche est de $O(m \log(m))$ ce qui a été montré comme étant le mieux que l'on puisse obtenir. Le second intérêt est qu'il a été possible d'obtenir des résultats sur des extensions utiles du problème comme minimiser le nombre de changements de fréquences ou considérer la puissance statique [GAU 05b].

4.3.2.2. *Ordonnancement FPP*

Contrairement au cas EDF, le problème d'ordonnancer de manière optimale vis-à-vis de l'énergie un ensemble de tâches sous FPP est NP-difficile comme prouvé dans [YUN 03]. Dans ce même article, Yun et Kim proposent un algorithme qui permet d'approcher la solution optimale de manière arbitrairement précise en temps polynomiale. Un peu plus tôt, dans [QUA 02], Quan et Hu proposent un algorithme optimal du point de vue énergie dont le point de départ est de remarquer que certains

ensembles de tâches particuliers peuvent être ordonnancés sous FPP à la vitesse optimale calculée pour EDF (voir paragraphe 4.3.2.1). Ensuite, en modifiant certaines échéances, il est toujours possible de se ramener à des ensembles de tâches ayant cette propriété et donc d'ordonnancer avec les vitesses calculées pour EDF. Néanmoins, cette approche est difficilement utilisable en pratique car sa complexité est supérieure à $O(m!)$ (voir [YUN 03]).

Nous choisissons de présenter dans la suite de ce paragraphe, une approche heuristique de Quan et Hu publiée dans [QUA 01], non optimale dans le cas général mais dont les performances sont supérieures à celles d'autres propositions présentées dans [SHI 99, SHI 00b]. Cet algorithme traite des ensembles de tâches non récurrentes mais, comme dans le cas EDF, il pourra être appliqué à des tâches périodiques en calculant la vitesse de chacune des instances mises à disposition pendant un ppcm des périodes.

4.3.2.2.1. Vitesse minimale pour une instance

L'idée de l'algorithme est la même que celle de [YAO 95] dans le cas EDF ; il s'agit d'identifier des intervalles de temps distincts sur lesquels on peut trouver une vitesse constante minimum qui garantisse la faisabilité. L'indice de toute tâche de l'ensemble $\mathcal{T} = \{\tau_1, \ldots, \tau_m\}$ indique sa priorité sous FPP avec la convention : « plus petite la valeur numérique, plus grande la priorité », et l'on considérera dans la suite que l'ensemble $\mathcal{T}$ est faisable à vitesse nominale.

Les auteurs introduisent un certain nombre de concepts et de notations ; en particulier, est appelé « τ_n-point d'ordonnancement » (*scheduling point*) tout instant qui est soit la date d'arrivée de τ_n ou une date d'arrivée d'une tâche plus prioritaire, soit D_n, l'échéance de τ_n. La vitesse de τ_n, notée S_n, ne dépendant dans leur stratégie que de certaines tâches plus prioritaires, on peut identifier un intervalle de recherche $[T_E(n), T_L(n)]$ tel que toutes les tâches plus prioritaires arrivées avant $T_E(n)$ ou après $T_L(n)$ n'interfèrent pas avec l'ordonnancement de τ_n. On fixe $T_L(n)$ à D_n alors que $T_E(n)$ est le plus grand τ_n-point d'ordonnancement t qui vérifie $t > A_i \Rightarrow t \geq D_i$ pour $i = 1..n$. L'intensité du travail à fournir entre les deux τ_n-points d'ordonnancement t_a et τ_b est définie comme :

$$I_n(t_a, t_b) = \frac{\sum_{i=1}^{n} C_i \cdot \mathbb{1}_{[t_a \leq A_i < t_b]}}{t_b - t_a}. \qquad [4.5]$$

Un intervalle $[t_a, t_b]$ est une « τ_n-période d'activité » (*busy interval*), avec τ_a, τ_b deux τ_n-points d'ordonnancement dans $[T_E(n), T_L(n)]$ tels que $t_a \leq A_n < t_b$, si le processeur est toujours utilisé dans $[t_a, t_b]$ lorsque la vitesse $I_n(t_a, t_b)$ est appliquée sur tout l'intervalle. Les auteurs montrent (lemme 3 de [QUA 01]) qu'un intervalle $[t_a, t_b]$ est une τ_n-période d'activité ssi $I_n(t_a, t) \geq I_n(t_a, t_b)$ pour tout t étant τ_n-point d'ordonnancement dans $]t_a, t_b]$. Parmi toutes les τ_n-période d'activité, la plus grande est appelée « l'intervalle essentiel » pour τ_n. Le résultat important est donné par le

lemme 4 de [QUA 01] qui dit que si $[t_s, t_f]$ est l'intervalle essentiel pour τ_n, alors, avec une vitesse processeur fixée à $I_n(t_s, t_f)$, l'exécution de τ_n sera nécessairement terminée avant D_n. Il est aussi montré que t_s et t_f vérifient :

$$I_n(t, t_s) < I_n(t_s, t_f) \quad \forall t \in [T_E(n), t_s[\qquad [4.6]$$

$$I_n(t_s, t_f) < I_n(t_f, t) \quad \forall t \in]t_f, T_L(n)]. \qquad [4.7]$$

Les deux propriétés [4.6] et [4.7] sont utilisées par l'algorithme de recherche de l'intervalle essentiel pour τ_n. Trouver S_n, la vitesse minimale qui garantit la faisabilité de τ_n consiste à déterminer $[t_s, t_f]$, l'intervalle essentiel de τ_n. L'algorithme proposé dans [QUA 01] construit itérativement cet intervalle en partant de A_n. Une recherche est effectuée à droite de A_n pour trouver t_1, le plus grand τ_n-point d'ordonnancement tel que $I_n(A_n, t_1) \leq I_n(A_n, t) \forall t \in [A_n, T_L(n)]$. Ensuite, on cherche à gauche de A_n pour trouver t_2, le plus petit τ_n-point d'ordonnancement tel que $I_n(t_2, t_1) > I_n(t, t_1) \forall t \in [T_E(n), A_n]$. A cette étape, l'intervalle essentiel courant est $[t_2, t_1]$; la recherche continue à droite de t_1 puis à gauche de t_2 pour trouver un éventuel intervalle essentiel plus grand que $[t_2, t_1]$. L'algorithme s'arrête lorsque deux intervalles essentiels correspondant à deux étapes successives de l'algorithme sont identiques. La complexité de cet algorithme est $O(m^2)$.

4.3.2.2.2. Ordonnancement global

La connaissance de l'intervalle essentiel et de la fréquence de fonctionnement associée, pour chacune des tâches, ne nous donne pas directement une solution d'ordonnancement faisable pour l'ensemble des tâches, en particulier parce que les intervalles essentiels peuvent se chevaucher. Dans [QUA 01], les auteurs montrent comment construire un ordonnancement global faisable avec une stratégie similaire à celle utilisée dans [YAO 95] pour EDF. L'intervalle essentiel avec la plus grande intensité de fonctionnement (voir l'équation [4.5]) est appelé l'intervalle critique de l'ensemble de tâches. Cet intervalle critique est noté $[t_s, t_f]$ et nous supposons qu'il s'agit à la base d'un intervalle essentiel pour τ_n. La première étape est d'allouer la vitesse $I_n(t_s, t_f)$ aux tâches associées à l'intervalle critique $[t_s, t_f]$, c'est-à-dire la tâche τ_n et les tâches de priorités supérieures à τ_n mises à disposition dans $[t_s, t_f[$. On construit ensuite un nouveau problème en supprimant l'intervalle déjà étudié et l'on détermine le prochain intervalle critique. Pour cela, il faut mettre à jour les dates des points d'ordonnancement plus grand que t_f en les diminuant d'une quantité $(t_f - t_s)$ et fixer tous les points d'ordonnancement restant entre $[t_s, t_f]$ à la valeur t_s. L'ensemble des intervalles critiques et les fréquences de fonctionnement associées constituent la solution au problème. La complexité d'une implémentation est $O(m^3)$ car il y a au plus m intervalles critiques nécessitant un calcul en $O(m^2)$ (voir le paragraphe 4.3.2.2.1).

4.4. Politiques dynamiques

Le plus souvent, dans les applications temps réel, toutes les instances des tâches ne nécessitent pas leur WCET. Compte tenu de la complexité des architectures matérielles actuelles (différents niveaux de caches, *pipelining*, etc.), il est même difficile d'évaluer de manière réaliste les WCETs (voir chapitre 5) et il est courant que ceux-ci soient surestimés et que donc aucune instance ne requiert son WCET. Certaines stratégies prennent en compte cette possible différence entre WCET et temps d'exécution effectif. On les appelle dans ce chapitre « politiques dynamiques » car les vitesses d'exécution utilisées varient en-ligne en fonction de l'historique de l'ordonnancement.

On distingue deux classes de techniques, la première appelée « ordonnancement stochastique » consiste à trouver, pour chaque cycle processeur, la vitesse qui minimise l'espérance de l'énergie en faisant certaines hypothèses probabilistes sur le temps d'exécution. La seconde classe de techniques, connue dans la littérature sous le terme de politiques « *gain reclaiming* » consiste à utiliser le temps processeur « économisé » par rapport au WCET pour réduire dans le futur la vitesse d'exécution d'une ou de plusieurs tâches.

4.4.1. *Ordonnancement stochastique*

En ordonnancement stochastique, l'exécution d'une tâche débute à une vitesse processeur faible et cette vitesse est augmentée graduellement au cours de l'exécution de la tâche de telle façon à respecter l'échéance si le WCET devait être requis. Comme le temps d'exécution est généralement plus petit que le WCET, les vitesses élevées ne sont le plus souvent pas utilisées d'où le gain en énergie. La façon dont la vitesse processeur varie au cours du temps est calculée hors-ligne en utilisant des informations probabilistes, obtenues par exemple à l'aide de mesures effectuées sur la plate-forme d'exécution, sur le nombre de cycles d'horloge requis pour finir une tâche. Nous présentons dans ce paragraphe les travaux de Gruian publiés dans [GRU 01, GRU 02] ; une approche similaire a été également proposée dans [LOR 01].

On note $F(x)$ la probabilité qu'une tâche finisse avant ou en le cycle d'instruction x ; en particulier $F(x) = 1$ pour $x \geq$ WCE où WCE est le nombre de cycles correspondant au WCET. L'espérance de l'énergie consommée est :

$$\overline{E} = \sum_{x=1}^{\text{WCE}} (1 - F(x)) \cdot e_x, \qquad [4.8]$$

où e_x est l'énergie consommée au cycle x. Le temps d'exécution du cycle x, notée k_x, correspond à la fréquence $f_x = 1/k_x$. Il est montré dans [GRU 02] que l'énergie consommée au cycle x est $e_x = \mathcal{K}\frac{1}{k_x^\beta}$ où $\mathcal{K}$ est une constante fonction du processeur et $\beta = 2/(\gamma - 1)$ soit $\beta = 2$ dans le cadre MOSFET classique (avec $\gamma = 2$,

voir paragraphe 4.2.1). La contrainte fixée est que la tâche doit terminer son exécution en un temps T, ce qui implique pour garantir la faisabilité dans tous les cas que $\sum_{x=1}^{\text{WCE}} k_x \leq T$. En remplaçant e_x par son expression dans [4.8], on obtient pour l'espérance $\overline{E} = \mathcal{K} \sum_{x=1}^{\text{WCE}} (1 - F(x))/k_x^2$. Il est prouvé dans [GRU 02] que $\overline{E}$ est minimisée en fixant le temps du cycle y à la valeur :

$$k_y = T \frac{\sqrt[3]{1 - F(y)}}{\sum_{x=1}^{\text{WCE}} \sqrt[3]{1 - F(y)}}.$$

En pratique, le nombre de cycles WCE est extrêmement grand et, pour réduire le nombre de valeurs à calculer et stocker, il est nécessaire de travailler sur des ensembles de cycles consécutifs. Se pose aussi le problème du nombre nécessairement fini de fréquences disponibles, ce qui peut changer la forme de la solution optimale. A notre connaissance, ces deux difficultés n'ont pas été entièrement résolues par les travaux existants. De notre point de vue, l'intérêt de l'ordonnancement stochastique reste donc aujourd'hui essentiellement théorique mais cette technique originale pourrait servir de base à des heuristiques efficaces.

4.4.2. *Redistribution du temps processeur non utilisé*

Le pessimisme d'une analyse de pire temps d'exécution a deux causes principales : la première est la surestimation du temps de chacun des « blocs de base » pris individuellement (voir chapitre 5 et [COL 03]), la seconde est que le programme n'empruntera pas nécessairement le chemin d'exécution le plus long. Il est possible de détecter en ligne et d'utiliser le fait que les temps d'exécution effectifs sont inférieurs aux WCETs. On distingue classiquement dans la littérature les approches où le temps CPU non utilisé par une tâche est réalloué à cette même tâche (*intra-task DVS*) et les approches où le temps est redistribué aux autres tâches du système (*inter-task DVS*). Ces techniques sont classiquement utilisées en complément des approches hors-ligne (voir section 4.3) utilisant les hypothèses de WCET.

4.4.2.1. *Redistribution intra-tâche*

Ces techniques requièrent l'insertion de points de mesure dans le code pour évaluer en-ligne le pire temps d'exécution restant et modifier en conséquence la vitesse de la tâche en cours. Le code peut être instrumenté par le préprocesseur du compilateur et l'on parle alors de *compiler-assisted speed scheduling*.

Le code d'une tâche est divisé en sections pour lesquelles le WCET est connu. La vitesse du processeur est recalculée en-ligne après chacune de ces sections en fonction de la différence entre le temps d'exécution effectif et le WCET. Plus l'exécution du programme progresse et plus la connaissance du temps d'exécution restant est précise,

et donc, plus la vitesse choisie sera proche de l'optimal. Il existe diverses stratégies (voir [MOS 00]) pour répartir au sein d'une tâche le temps qui a été détecté non utilisé : l'intégralité peut être allouée à la prochaine section où l'on peut répartir entre les différentes sections qui suivent, par exemple proportionnellement aux pires temps d'exécution prévus.

Une difficulté majeure de ces approches est de choisir la bonne granularité d'instrumentation. En effet, le temps de commutation de fréquence et les instructions supplémentaires rajoutées pour la mesure des temps d'exécution ainsi que le re-calcul de la vitesse processeur peuvent induire une surcharge supérieure aux gains potentiels. A notre connaissance, ce problème délicat a été encore incomplètement traité par les travaux existants dans le domaine [MOS 00, SHI 01a, SHI 01b] ou, plus récemment, [KUM 05].

A noter qu'une évaluation comparative, en termes de performances et de facilité de mise en oeuvre, des techniques de redistribution intra-tâche et d'ordonnancement stochastique, peut être trouvée dans [GRU 02].

4.4.2.2. *Redistribution inter-tâches*

Au contraire des techniques de redistribution intra-tâche, les techniques inter-tâches ne nécessitent pas d'instrumentation du code. Le principe est que lorsqu'une tâche se termine, le temps d'exécution qu'elle n'a pas utilisé est redistribué à la ou les tâches suivantes.

Une étude importante est [PIL 01] qui propose des solutions simples et efficaces pour des tâches à échéances égales aux périodes sous les politiques EDF et FPP. Supposons qu'une tâche τ_k, de WCET (en nombre de cycles processeurs) C_k, termine l'exécution d'une de ses instances en utilisant $cc_k < C_k$ cycles. L'idée de la stratégie *Cycle-Conserving EDF* est d'utiliser ce temps gagné pour réduire localement la vitesse de toutes les autres instances actives jusqu'à l'arrivée de la prochaine instance de τ_k. Le calcul se fait à l'aide des tests de faisabilité présentés aux paragraphes 4.3.1.1.1 et 4.3.1.2.2 avec l'utilisation processeur de τ_k égale à cc_k et non plus C_k. A la prochaine arrivée d'une instance de τ_k, il faudra re-calculer la vitesse sous les hypothèses pessimistes de WCET.

Dans le contexte d'EDF, les même auteurs proposent une politique qui spécule sur les gains à venir (on parle dans la littérature de *speculative speed reduction*). Cette politique, appelée *Look-Ahead EDF*, consiste à ne faire à chaque instant que le minimum de travail qui ne mettent pas en péril la faisabilité du système dans le futur. Ainsi, à court terme, les fréquences processeur élevées ne sont pas utilisées et s'il advient que les tâches utilisent beaucoup moins que leur WCET, ces fréquences élevées ne seront jamais utilisées ; *Look-Ahead EDF* est alors beaucoup plus performant que *Cycle-Conserving EDF*. D'autres approches efficaces ont été développées par la suite, le lecteur pourra consulter en particulier [AYD 04].

4.5. Conclusions

Nous avons fait dans ce document un tour d'horizon des stratégies d'ordonnancement sous contraintes de temps et d'énergie en présentant quelques travaux que nous jugions importants, parce que ce sont des résultats de base ou par leur efficacité pratique. Nous avons distingué les approches hors-ligne des approches dynamiques, dont les représentants les plus utiles en pratique sont de la classe des politiques à « redistribution du temps processeur non utilisé ». Approches hors-ligne et dynamiques ne sont pas antagonistes et une efficacité maximale sera généralement obtenue en couplant les deux. Le lecteur intéressé pourra par exemple consulter [AYD 04] pour une illustration.

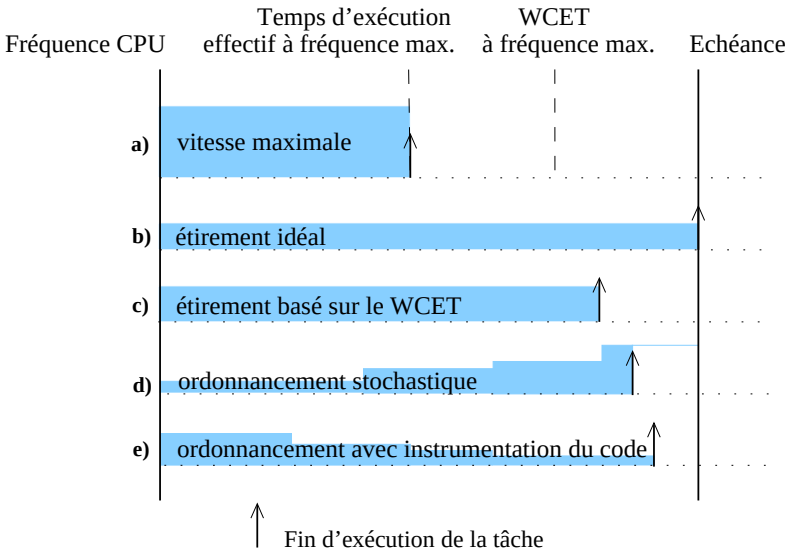

Figure 4.2. *Profils de consommation d'une même tâche sous différentes stratégies d'ordonnancement (d'après [GRU 02])*

Nous concluons sur le schéma 4.2 adapté de [GRU 02] qui permet de mettre en perspective la plupart des approches existantes. Cette figure représente l'évolution de la fréquence CPU pendant l'exécution d'une tâche sous différentes stratégies d'ordonnancement faible-consommation. Le WCET de la tâche, exprimé ici en nombre de cycles processeur, est connu mais comme c'est le cas le plus généralement en pratique, l'exécution considérée pour la figure 4.2 nécessite moins de cycles processeur que le WCET.

La première stratégie (cas a) sur la figure 4.2) est d'exécuter les tâches à la vitesse nominale (c'est-à-dire maximale) du processeur ce qui est le moins efficace d'un

point de vue énergétique. Si l'on connaît *a priori* le nombre précis de cycles processeur utilisés par la tâche, il est alors possible de diminuer la vitesse de cette tâche de façon optimale (cas b)). Cette solution, appelée « étirement idéal », est en pratique non utilisable car, dans le cas général, on ne peut naturellement prédire à l'avance le nombre de cycles CPU requis par une tâche. Néanmoins, cette stratégie fournit un bon référentiel pour évaluer *a posteriori* (après exécution ou simulation) les performances de stratégies d'ordonnancement faible-consommation. Le cas c) représente « l'étirement basé sur le WCET » qui est d'autant moins efficace d'un point de vue énergétique que les tâches ne consomment pas l'intégralité de leur WCET. Dans ce cas de meilleures solutions existent : « l'ordonnancement stochastique » (cas d), voir paragraphe 4.4.1) ou « l'ordonnancement avec instrumentation de code » (cas e), voir paragraphe 4.4.2.1) qui ont des profils de consommation opposés. En effet, la vitesse du processeur augmente avec le temps en ordonnancement stochastique alors qu'elle diminue avec l'instrumentation de code au fur et à mesure que la connaissance du temps d'exécution réel restant devient plus précise.

Notons que si la figure 4.2 permet d'appréhender comment certaines stratégies d'ordonnancement influent sur le profil de consommation, le problème global de l'ordonnancement faible consommation est plus complexe car les systèmes sont généralement multi-tâches, et les différentes activités sont en concurrence pour le processeur. Comme nous l'avons vu dans ce chapitre, le choix de la fréquence de fonctionnement d'une tâche, ou d'une instance de tâche, doit considérer la politique d'ordonnancement sous-jacente et les autres activités en concurrence.

4.6. Perspectives : vers une prise en compte plus fine du matériel

Les études dans le domaine de l'ordonnancement faible consommation ont pris leur essor un peu après le début des années 1990, motivées par les besoins naissant de l'industrie. Depuis plusieurs années, on assiste à un nombre de publications et d'implémentations très considérables dans ce domaine. Jusqu'à maintenant, la plupart des études ont fait de fortes hypothèses simplificatrices sur les technologies matérielles sous-jacentes, en particulier le fait de négliger le courant de fuite du CPU et de considérer des modèles de batterie simplistes. Sous ces hypothèses sur le *hardware*, il existe des résultats optimaux, ou proches de l'optimal, pour la quasi-totalité des besoins applicatifs. Le challenge en ordonnancement faible-consommation est maintenant, de notre point de vue, de prendre en compte plus finement le matériel.

Ainsi, le courant de fuite, qui pouvait être négligé il y a encore quelques années, devient maintenant très significatif avec les dernières technologies des semiconducteurs [Sem 05]. Des travaux dans cette direction ont déjà été effectués, par exemple [JEJ 04] pour EDF et [QUA 04] pour FPP, mais beaucoup reste à faire. Techniquement, la meilleure façon de réduire le courant de fuite est de mettre le processeur

dans un état de veille. Au contraire, réduire la consommation due à la puissance dynamique par diminution de fréquence est le plus efficace lorsque le processeur fonctionne le plus longtemps possible à vitesse réduite (conséquence de la convexité de la puissance dynamique dissipée en fonction de la fréquence, voir 4.2.1). L'optimum passe donc par un compromis qui nécessite généralement de repenser en profondeur les solutions existantes.

Jusqu'à maintenant, la quasi-totalité des études en ordonnancement faible consommation ont considéré un modèle de batterie idéale où le profil de décharge (c'est-à-dire la variation de l'intensité fournie au cours du temps) n'influerait pas sur la quantité d'énergie totale qu'il est possible d'extraire de la batterie, et donc sur le temps de fonctionnement du système. Des expérimentations, par exemple dans [RAO 05], ont montré qu'en pratique le profil de décharge influait significativement sur la charge qu'il est possible d'extraire d'une batterie, ce qui peut être pris en compte par les solutions d'ordonnancement. Ces techniques sont référencées dans la littérature sous le terme de *Battery Aware Scheduling* (le lecteur intéressé pourra consulter [RAO 06] pour un point d'entrée dans le domaine) et se fondent généralement sur des heuristiques tirées de l'étude de modèles de batterie, par exemple des modèles stochastiques à gros grain [RAO 05] ou des modèles simulant précisément les réactions electro-chimiques internes aux batteries [RAK 03]. Ainsi, il a été montré dans [RAK 03] qu'un profil de décharge décroissant maximisait l'énergie qui peut être fournie par une batterie et des heuristiques efficaces utilisant cette propriété ont été récemment proposées dans des contextes applicatifs variés [ZHU 05, Y.C 05, RAO 06]. Si les gains sont significatifs dans les études précitées, celles-ci ne couvrent pas tous les besoins et beaucoup reste encore à faire.

Enfin, l'ordonnancement processeur doit s'étudier davantage en considérant les autres composants du système : mémoires et périphériques comme ASIC (*Application Specific Integrated Circuit)*) et DSP (*Digital Signal Processor*). Par exemple, réduire la fréquence CPU peut augmenter le temps d'utilisation des périphériques et donc leur consommation [KIM 01]. Là encore, des compromis sont à trouver entre DVS au niveau du processeur et DPM (mode veille) au niveau des périphériques. Comme cela a été fait dans [BIN 05], il faut également intégrer le fait que les entrées/sorties avec les périphériques sont généralement d'une durée fixe et que le temps d'exécution de certaines parties du code est donc presque indépendant de la fréquence de fonctionnement du processeur.

Et l'on voit que les paramètres à prendre en compte lors de la conception ou du choix d'une stratégie d'ordonnancement faible-consommation sont multiples et requiert souvent des solutions antagonistes. Le bon choix pour une architecture matérielle et une application donnée relèvera généralement de compromis délicats à trouver. Le challenge est maintenant aussi de développer les outils méthodologiques et logiciels qui aideront le concepteur d'un système dans la définition de ses stratégies d'ordonnancement.

4.7. Bibliographie

[ABN 98] ABNOUS A., SENO K., ICHIKAWA Y., WAN M., RABAEY J., « Evaluation of a Low-Power Reconfigurable DSP Architecture », *IPPS/SPDP Workshops*, p. 55–60, 1998.

[ANC 03] ANCEAU F., « Une Technique de Réduction de la Puissance Dissipée par l'Horlogerie des Circuits Complexes Rapides », *4ième journée d'études Faible Tension Faible Consommation (FTFC'03)*, p. 125-130, mai 2003.

[AYD 01] AYDIN H., R. M., MOSSÈ D., P. M.-A., « Determining Optimal Processor Speeds for Periodic Real-Time Tasks », *Euromicro Conference on Real-Time Systems*, p. 225-232, 2001.

[AYD 04] AYDIN H., MELHEM R., MOSSÉ D., MEJIA-ALVAREZ P., « Power-Aware Scheduling for Periodic Real-Time Tasks », *IEEE Transactions on Computers*, vol. 53, n°5, p. 584-600, 2004.

[BEN 00] BENINI L., BOGLIOLO A., MICHELI G. D., « A Survey of Design Techniques for System-Level Dynamic Power Management », *IEEE Transactions on Very Large Scale Integration (VLSI) Ssystems*, vol. 8, n°3, p. 299-316, juin 2000.

[BIN 05] BINI E., BUTTAZZO G., LIPARI G., « Speed Modulation in Energy-Aware Real-Time System », *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, juillet 2005.

[BRI 04] BRITO R., NAVET N., « Low-Power Round-Robin Scheduling », *12th International Conference on Real-Time Systems (RTS'04)*, mars 2004.

[COF 76] COFFMAN E., « Introduction to Deterministic Scheduling Theory », COFFMAN E., Ed., *Computer and Job-Shop Scheduling Theory*, Wiley, New York, 1976.

[COL 03] COLIN A., PUAUT I., ROCHANGE C., SAINRAT P., « Calcul de Majorants de Pire Temps d'Exécution : Etat de l'Art », *Technique et Science Informatiques*, vol. 22, n°5, p. 651–677, 2003.

[COM 02] COMPAQ, INTEL, MICROSOFT, PHOENIX, TOSHIBA, Advanced Configuration and Power Interface Specification, octobre 2002, revision 2.0b.

[DAV 03] DAVID R., SENTIEYS O., « Architectures reconfigurables : opportunités pour la faible consommation », *4ième journée d'études Faible Tension Faible Consommation (FTFC'03)*, p. 31-40, mai 2003.

[GAU 05a] GAUJAL B., NAVET N., « Yao et al's Algorithm Revisited », *soumis à Real-Time Systems, version préliminaire disponible sous forme de rapport de recherche INRIA RR-5125*, 2005.

[GAU 05b] GAUJAL B., NAVET N., WALSH C., « Shortest Path Algorithms for Real-Time Scheduling of FIFO tasks with Optimal Energy Use », *ACM Transactions on Embedded Computing Systems*, vol. 4, n°4, p. 907–933, novembre 2005.

[GRU 01] GRUIAN F., « On energy reduction in hard real-time systems containing tasks with stochastic execution times », *IEEE Workshop on Power Management for Real-Time and Embedded Systems*, p. 11–16, 2001.

[GRU 02] GRUIAN F., Energy-Centric Scheduling for Real-Time Systems, PhD thesis, Lund Institute of Technology, 2002.

[Int 03] INTEL CORPORATION, Intel 80200 Processor based on Intel XScale Microarchitecture Datasheet, Number : 273414-005, janvier 2003.

[JEJ 04] JEJURIKAR R., PEREIRA C., GUPTA R., « Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems », *DAC '04 : Proceedings of the 41st annual conference on Design automation*, New York, ACM Press, p. 275–280, 2004.

[KIM 01] KIM M., HA S., « Hybrid Run-time Power Management Technique for Real-time Embedded System with Voltage Scalable Processor », *Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems (LCTES'01)*, New York, ACM Press, p. 11–19, 2001.

[KUM 05] KUMAR G. S. A., MANIMARAN G., « An Intra-Task DVS Algorithm Exploiting Program Path Locality for Real-time Embedded Systems », *Proc. IEEE Intl. Conf. on High Performance Computing (HiPC)*, 2005.

[LEH 89] LEHOCZKY J., SHA L., DING Y., « The Rate Monotonic Scheduling Algorithm : Exact Characterization and Average Behavior », *Proc. of the IEEE Real-Time Systems Symposium*, 1989.

[LIU 73] LIU C., LAYLAND J., « Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment », *Journal of the ACM*, vol. 20, n°1, p. 40-61, février 1973.

[LOR 01] LORCH J., SMITH A., « Improving Dynamic Voltage Scaling Algorithms with PACE », *ACM SIGMETRICS 2001 Conference*, p. 50–61, 2001.

[MOS 00] MOSSÈ D., AYDIN H., CHILDERS B., MELHEM R., « Compiler-Assisted Dynamic Power-Aware Scheduling for Real-Time Applications », *Workshop on Compiler and Operating Systems for Low-Power*, 2000.

[PAR 00] PARAIN F., BANÂTRE M., CABILIC G., HIGUERA T., ISSARNY V., LESOT J.-P., Techniques de Réduction de la Consommation dans les Systèmes Embarqués Temps-Réel, Rapport n°3932, IRISA, mai 2000.

[PIL 01] PILLAI P., SHIN K., « Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems », *ACM Symposium on Operating Systems Principles*, p. 89-102, 2001.

[POU 01] POUWELSE J., LANGENDOEN K., SIPS H., « Dynamic voltage scaling on a low-power microprocessor », *Proceedings of the 7th annual international conference on Mobile computing and networking (MobiCom'01)*, New York, NY, USA, ACM Press, p. 251–259, 2001.

[QUA 01] QUAN G., HU X., « Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors », *Design Automation Conference*, p. 828-833, 2001, Version étendue disponible sous forme de rapport de recherche de l'université de Notre Dame (USA).

[QUA 02] QUAN G., HU X., « Minimum Energy Fixed-Priority Scheduling for Variable Voltage Processors », *Design, Automation and Test in Europe Conference and Exhibition (DATE'02)*, p. 782-787, 2002.

[QUA 04] QUAN G., NIU L., HU X., MOCHOCKI B., « Fixed Priority Scheduling for Reducing Overall Energy on Variable Voltage Processors », *Proc. of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, p. 309–318, 2004.

[RAB 96] RABAEY J., PEDRAM M., Eds., *Low Power Design Methodologies*, Kluwer Academic Publishers, 1996.

[RAK 03] RAKHMATOV D., VRUDHULA S., « Energy Management for Battery-Powered Embedded Systems », *ACM Transactions on Embedded Computing Systems*, vol. 2, n°3, p. 277–324, 2003.

[RAO 05] RAO V., SINGHAL G., KUMAR A., NAVET N., « Battery Model for Embedded Systems », *Proc. of the 18th International Conference on VLSI Design (VLSI'2005)*, p. 105–110, janvier 2005.

[RAO 06] RAO V., SINGHAL G., NAVET N., KUMAR A., VISWESWARAN G., « Battery Aware Dynamic Scheduling for Periodic Task Graphs », *Proceedings of the 14th International Workshop on Parallel and Distributed Real-Time Systems 2006 (WPDRTS 2006)*, avril 2006, à paraître.

[REM 03] REMOND Y., SIRIANNI A., SICARD G., RENAUDIN M., « Estimation et Optimisation de la Consommation d'Energie des Circuits Asynchrones », *4ième journée d'études Faible Tension Faible Consommation (FTFC'03)*, p. 59-64, mai 2003.

[SAE 03] SAEWONG S., RAJKUMAR R., « Practical Voltage-Scaling for Fixed-Priority Real Time Systems », *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS'03)*, p. 127-132, 2003.

[SAL 03] SALHIENE M. E., FESQUET L., RENAUDIN M., « Adaptation Dynamique de la Puissance des Systèmes Embarqués : les Systèmes Asynchrones Surclassent les Systèmes Synchrones », *4ième journée d'études Faible Tension Faible Consommation (FTFC'03)*, p. 51-58, mai 2003.

[Sem 05] SEMICONDUCTOR INDUSTRY ASSOCIATION, « International Technology Roadmap for Semiconductors », Disponible à l'url http ://public.itrs.net/, 2005.

[SHI 99] SHIN Y., CHOI K., « Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems », *Design Automation Conference*, p. 134–139, 1999.

[SHI 00a] SHIN Y., CHOI K., System-Level Power Optimization of Embedded Systems, Rapport n°SNU-EE-TR-2000-3, School of Electrical Engineering, Seoul National University, 2000.

[SHI 00b] SHIN Y., CHOI K., SAKURAI T., « Power Optimization of Real-Time Embedded Systems on Variable Speed Processors », *International Conference on Computer Aided Design*, p. 365-368, 2000.

[SHI 01a] SHIN D., KIM J., LEE S., « Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications », *IEEE Design & Test of Computers*, vol. 18, n°2, 2001.

[SHI 01b] SHIN Y., KAWAGUCHI H., SAKURAI T., « Cooperative Voltage Scaling (CVS) between OS and Applications for Low-Power Real-Time Systems », *IEEE Custom Integrated Circuits Conference*, p. 553-556, 2001.

[STA 98] STANKOVIC J., SPURI M., RAMAMRITHAM K., BUTTAZZO C., *Deadline Scheduling for Real-Time Systems*, Kluwer Academic Publishers, Norwell, USA, 1998.

[SU 95] SU C.-L., DESPAIN A., « Cache Designs for Energy Efficiency », *28th Hawaii International Conference on System Science*, p. 306–315, 1995.

[TIW 94] TIWARI V., MALIK S., WOLFE A., « Power Analysis of Embedded Software : a First Step towards Software Power Minimization », *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, n˚4, p. 437–445, 1994.

[WEI 94] WEISER M., WELCH B., DEMERS A. J., SHENKER S., « Scheduling for Reduced CPU Energy », *First Symposium on Operating Systems Design and Implementation*, p. 13-23, 1994.

[YAO 95] YAO F., DEMERS A., SHENKER S., « A Scheduling Model for Reduced CPU Energy », *Proceedings of lEEE Annual Foundations of Computer Science*, p. 374–382, 1995.

[Y.C 05] Y.CHAI, REDDY S., POMERANZ I., AL-HASHIMI B., « Battery Aware Dynamic Voltage Scaling in Multiprocessor Embedded System », *IEEE International Conference on Circuits and Systems (ISCAS 2005)*, mai 2005.

[YUN 03] YUN H.-S., KIM J., « On Energy-Optimal Voltage Scheduling for Fixed-Priority Hard Real-time Systems », *ACM Transactions on Embedded Computing Systems*, vol. 2, n˚3, p. 393–430, août 2003.

[ZEN 02] ZENG H., FAN X., ELLIS C., LEBECK A., VAHDAT A., « ECOSystem : Managing Energy as a First Class Operating System Resource », *Proceedings of the tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, octobre 2002.

[ZHU 05] ZHUO J., CHAKRABARTI C., « An Efficient Dynamic Task Scheduling Algorithm for Battery Powered DVS Systems », *Asia and South Pacific Design Automation Conference (ASPDAC'05)*, 2005.